DYNAMIC MULTI THRESHOLD PRIORITY JOB SCHEDULING

A. SHAJI GEORGE[1], Dr. K. KRISHNAMOORTHY [2]

[1] Research Scholar, Faculty of Computing & Information Technology, Himalayan University, Itanagar, Papumpare- 791111, Arunachal Pradesh, India

[2]Professor, Department of Computer Science & Engineering, Sudharsan Engineering College, Pudukkottai-622003,Tamilnadu, India

ABSTRACT

In this paper, we propose new dynamic priority job scheduling algorithm and integrate with Wireless Sensor Networks to improve throughput of the network. Our proposed Dynamic Multi Threshold Priority Job scheduling algorithm ensures a decrease in loss delay for the lower priority level service with acceptable fairness towards higher priority level data. Threshold algorithm is compared with the commonly used scheduling algorithms such as First-Come-First-Serve (FCFS) and fixed priority non-preemptive. Simulation results illustrate that the Dynamic Multi Threshold Priority Job scheduling algorithm can provide a better QoS for low priority jobs while keeping the QoS levels for high priority jobs at similar levels. Our new job scheduler can effectively mimic the emergency job first scheduling policy. To demonstrate its performance. Our experimental and simulation results have strongly confirmed the effectiveness of our design: our new job scheduler can reduce the delay and waiting time.

Introduction

In this chapter, we introduce the concept of wireless sensor networks (WSN). We present the main fields of application, then we provide an overview of sensor networks process. Finally, we review research challenges for WSN.A new class of networks has appeared in the last few years: the wireless sensor networks. These networks consist of individual sensor nodes deployed in a given area that cooperatively collect and carry data to a main entity in order to monitor physical or environmental conditions. The main entity, also denoted as base station or sink, can be connected to an infrastructure or to the Internet through a gateway. The system can also operate without the need for an existing infrastructure, thus the network operates autonomously. The user can periodically collect the

gathered data through a direct connection to the sink using a mobile device, such as a laptop or a smart phone.

Job Scheduling

Job scheduling plays an important role in improving the overall system performance in wireless sensor networks. Simple job scheduling policies, such as Fair and FIFO scheduling, do not consider job sizes and may degrade the performance when jobs of varying sizes arrive. More elaborate job scheduling policies make the convenient assumption that jobs are recurring, and complete information about their sizes is available from their prior runs. In this paper, we design and implement an efficient and practical job scheduler for wireless sensor processing systems to achieve better performance even without prior information about job sizes. In Wireless Sensor Networks, it is needed to schedule different types of jobs such as small and larger. It is important to reduce delay in job queue over the network. Scheduling is the strategy by which the system decides which task should be executed at any given time. Generally, it is used for load balance and system resources to be served effectively and with desired quality. Scheduling algorithms are classified as preemptive and non-preemptive. In preemptive scheduling, arrival of high priority data suspends current lower priority data. In non-preemptive scheduling, the current lower priority data is processed until it is completed.

Preemptive schedulers provide better performance for system utilization. Non-preemptive schedulers encounter the starvation problem. Non-preemptive schedulers may give better performance and robustness on reduced systems. The major scheduling algorithms for WSNs are FCFS and fixed priority non-preemptive scheduling. In FCFS algorithm, each priority level of data's loss ratio results are assumed to be the same since priority is not criteria in this algorithm. In fixed priority non-preemptive scheduling algorithm, lower priority level data can get lost excessively and the higher priority level data are processed more quickly and there is a definite unfairness between priority levels.

In wireless sensor network there are many design issues, such as transmission delay, routing and data aggregation and reducing sensor energy consumption, job scheduling and fairness among node latency should be minimum. The sensor node sends higher priority data from the high priority queue first and if no data are present in high priority queue, nodes sends low priority data from low priority queue. The concept of priority data transmission guaranties quicker data transmission of high priority data in contrast to low-priority data.

In this work, we develop a new scheduling algorithm for WSNs which can provide a better QoS and resolve the disadvantages of the current algorithms. The proposed Dynamic Multi Threshold Priority Job scheduling algorithm tries to maximize the level of QoS provided to different tasks by reducing the delay and waiting times. Also, the proposed scheduling algorithm is non-preemptive due to structure of reduced system load of WSNs.

Literature Review

Rudraneel et al [1] presented a starvation free priority based resource scheduling technique for a multitenant storm cluster. A prototype proof of concept framework to implement the technique and associated algorithms is described. An initial performance evaluation of the proposed scheduler on a small system with 3 topologies is presented for both the proposed as well as Isolation Scheduler.

Tim et al [2] evaluated a generic mechanism to evaluate the performance of a schedule, regarding interference and conflicts, to rank the schedule compared to other possible ones. At the core of this mechanism lies the interference graph. This is a transformation of a schedule and topology to a graph that represents the possible interference occurring in the scheduled network. Then, schedules are evaluated based on the weighted density of their corresponding interference graph.

Zhiming et al [3] designed and implemented a new job scheduler, called LAS_MQ, to utilize a multilevel priority queue to mimic a shortest job first policy without complete prior information of jobs. They also proposed a new way to obtain the amount of service that the job would receive in each stage, as well as a new policy for job scheduling in each queue to improve the performance.

Zhaocong et al [4] introduced the YARN hybrid scheduling strategy, introduces the YARN framework and the three most commonly used schedulers FIFO, Capacity and Fair, then introduces the job composition and queue configuration. Finally, the experimental analysis is carried out. For the interactive job, Fair scheduler performance is better than the Capacity scheduler; for batch jobs, resources in the unrestricted conditions, Fair scheduler performance is better than the Capacity scheduler. However, due to the limitations of cluster resources, this experiment cannot submit too many large jobs at the same time, we can expand the cluster, re-test batch jobs.

Anilkumar et al [5] presented a preemptive job scheduler based on a 3-layer Back propagation Neural Network (BPNN) and a greedy task alignment procedure. The BPNN estimates priority values of jobs based on the attributes of their subtasks and the given job

selection criteria of the scheduler. The scheduler is formulated in such a way that, at each time interval, the most priority job will be selected from the job queue before the next job arrives. The selected job is only preempted by a new job if its priority is less than the new job and then the preempted job will be restarted when its priority reaches high. When a predefined threshold time is reached, the job queue is refreshed to eliminate the old and low priority jobs. The proposed satisfiability measure based on job validation test, BPNN convergence test and cost value assure the efficiency of the scheduler.

Methodology

In this paper, we plan and developed an effective and practical job scheduler in data processing frameworks, without any consideration of time of the job sizes. A multi-level job queue in our design is one of the advantage, which is mainly used for separating short jobs from large jobs in an efficient manner and to imitate continuous sharing for jobs with similar sizes because eventually these jobs will enter the same queue and it will be arranged in a sequence order in each queue. We have to create a design in an efficient way to schedule jobs in the same queue and across different queues to improve the performance.

A new job scheduling algorithm has proposed in this project and the Wireless Sensor Networks are integrated with it to increase the end to end delay reduction. In the queue to reduce the waiting time for the lower priority level data with acceptable fairness towards higher priority level data, the Dynamic Multi Threshold Job Scheduling algorithm has mainly developed for achieving this mechanism. The commonly used scheduling algorithms like First-Come-First-Serve (FCFS) and fixed priority non-preemptive are compared with Threshold algorithm. A better QoS provides for low priority jobs while maintaining the QoS levels for high priority jobs at homogeneous levels in the Dynamic Multi Threshold Priority job scheduling algorithm that are illustrated in the simulation results.

DYNAMIC MULTI THRESHOLD PRIORITY JOB SCHEDULING ALGORITHM

A. Overview

The Dynamic Multi Threshold Priority Job Scheduling (DMTJS) algorithm is developed for this work. According to their priorities jobs are placed into the waiting queue in DTMPS algorithm and each priority level has its own threshold value.

The main concept of the DMTJS algorithm is to ordering the higher priority jobs and lower priority jobs according to their threshold values. Locating the lower priority jobs and replacing with higher priority arrival jobs at appropriate own threshold value is a most important function of DMTJS algorithm.

Two priority scheduling's are described previously, for larger jobs the low priority scheduling are used and for smaller jobs the high priority scheduling are used. The multiple queues are included in the low and higher priority schedulers. Only after completion of all the jobs from the first queue to the ($i-1$)-th queue, the jobs in the i-th queue get started and FIFO is used in each queue. This queue completion root a long waiting time to progress a jobs in the queue. To overcome this problem by executes the dynamic jobs in the queue, we have designed an additional priority scheduler approach.

An extra scheduler is called as a medium priority scheduler which is specially developed to handle the emergency jobs in the queue. Those emergency jobs will be processed first, which jobs are assigned to the medium priority scheduler. The services of the jobs are assigned to the immediate execution and which is based on the threshold value. The waiting queue contain various priority threshold values. The dynamic threshold value allocation and the dynamic priority scheduler process are our main works.

B. Design

For each priority level the threshold values are described with various values. The high, medium and the low are the three levels of Priority levels. The priority and assignment of job scheduling operation are properly allocated for each jobs and those are similar with fixed priority non-preemptive scheduling algorithm.

Priority – 1 (The high priority level): %k of capacity in the waiting queue.

Priority – 2 (The medium priority level): %m of capacity in the waiting queue.

Priority – 3 (The low priority level): %n of capacity in the waiting queue.

Threshold values are specified as n> m > k. Depending on the traffic situation in the network these values got changed.

The own threshold value of high priority level jobs is described as k and it must be lower than own threshold values of lower priority level job which are m and n. Fairness of higher priority level job towards the lower priority level data so it should have a lower value. Every priority level has a unique threshold level. If higher priority data would not get assigned to threshold level, the higher priority levels' assigned threshold level can varying with lower priority level's assigned threshold level.

If the front of proper threshold is full and lower priority job is found, the higher priority has changed with it in DMTJS algorithm. The jobs which having higher priority are not just arranged with allocated threshold level. Due to the higher priority job assigned threshold, if there is no space, it has been checking at lower threshold levels.

Priority-1 (the high priority level) jobs are appeared and there is no place at assigned threshold (Threshold-1), then it can be checked at lower threshold (Threshold-2) then get into the waiting queue. By using their own threshold the higher priority jobs (Priority 1 and Priority 2) are checked itself and if there is no place to get in, they used the lower threshold levels. The higher priority level's changing allocated threshold levels which provide that higher priority level jobs get enter to the waiting queue. In the priority queues, the dynamic job thresholds are helpful for decrease the waiting time.

Result

In this part, we examine the proposed Dynamic Multi Threshold Job Scheduling scheme's performance into different existing scheduling methods. For the purpose of calculating the performance of the proposed DMTJS, we compare it opposed to the FCFS, and Multilevel Queue scheduling schemes, to check the end-to-end delay of various types of traffic in the ready queues of active nodes. Each node in the ready queue of can hold a maximum of 50 tasks. For identifying the task type, each task have unique Type ID. For example, consider type 0 is a real-time task. Based on the emergency and processing time of the task, the emergency jobs are placed into the ready queue. Also, the services are randomly assigned to every jobs, and we placing the jobs into the highest-priority queue which have the smaller size. We calculate the average end-to-end delay of transmitting multiple various priority jobs to the base station (BS) in the following. Again, to represent the jobs that are sensed at a sensor node we had use the task and data in an interchangeable way.

Real-time Priority 1 Queue Data

Let us consider that a node x, residing at level lk is sensing a real time, emergency event, e.g., fire detection. This node transfer the emergency priority 1 data to BS by using lk−1 intermediate levels. We use the following mechanism by a real-time jobs reaches a neighboring active node at every time, y in an upper Level, a non-real time lower priority data is get executed at that node. To send real-time data, the data delivery at y is preempted.

Transmission delay or time that is needed to place a real time data into the medium is equal to datapr1 /st from a node. To transmit data from the source to destination, the propagation time or delay can be calculated as d /sp. The end-to-end delay for sending a real time data satisfies the following inequality by considering the methodology which is mentioned above.

$$delay_{p1} \geq lk \times \left(data_{p1/St} + p1_{proc}(t)\right) + d/Sp + (lk \times t_{over\,head}) \quad \text{------ (1)}$$

Where $delay_{p1}$=The real time data size

St=Transmission speed

d=the distance from Source node to BS

where d= $\sum_{i=1}^{lk}(di)$

Sp= The propagation speed among the wireless medium, pr1proc(t)= The processing time of real time tasks at each node, $t_{overhead}$in terms of context switching and queuing time (for preemption). If there is a number, a real time task t1 want to wait, np1 of real including time task ahead of t1 at the pr1 queue. Just assume, the same size is used for all real time data. Therefore, the end -to- end delay for real time task t1considering that t1 has npr1 number of real time task ahead of it,

$$\text{delay t1} \geq \sum_{j=1}^{np\,1}(delayp1)i \quad \text{-------(2)}$$

Non-real time Priority 2 Queue Data

Tasks at p2 queue can be preempted by real-time ones. We first contemplate the outline when a real-time task is sensed at node 11 and is forwarded to BS by using the relay nodes 9, 6, and 2. It must be observed that tasks are available at the pr2 queue at nodes 9, 6 and 2. Since one real-time task is available at the pr1 queue of nodes 9, 6, and 2, real-time tasks will be processed and transmitted first during the timeslot of nodes 9, 6, and 2. The pr2 tasks are processed in the remaining time of the timeslots.

When a new job is submitted to the system, it should first go through the job admission module, we can see this in our overall design. The total number of running jobs only have controlled by us in our case because number of jobs are running there which may cause hanging. The job is admitted and queued for its share of resources which have a small amount of running jobs.

By updating the configuration file on a real-time basis, the capacity scheduler can change the capacities of queues. In our development, each application is allocated to an individual queue. Then by setting the capacities of queues, we can handle the amount of resources for each application. The new job will be submitted to the queue and start running if the number of containers assigned to the application is above zero, after scheduling. We keep watching the job's running status that are task completion events and stage progresses during the lifetime of the job, which are the parts of the inputs for our scheduler. The job admission module has monitored when a job gets completed and it also verify whether all the jobs got completed, if not it will submit more jobs to the scheduler.

The unit of allocation is one core and 2 GB of main memory and it is a container in the scheduler. The total amount of memory helps to fix and bound the total number of

containers in the cluster in this case. There is a problem on scheduling is how to place the jobs to those containers. During the time of scheduling process to allocate a proper number of containers, we should keep tracking of the number of remaining tasks in the current stage of the job.

The very important thing of the job is acquiring the number of remaining tasks in the current stage. By using the total number of tasks minus the number of successfully completed tasks, we can estimate the number of tasks in our deployment. Anyway, the first main problem is how to get the total number of tasks for a job. We can set the tasks total amount, but the real total amount of tasks could be varied from the values in the arrangement. By investigating the number of splits of the inputs after the job is submitted to the queue and looking for the total number of job services we can solve this issue.

For investigation, we took 2 dissimilar parameters "time to completion" and "throughput" Figure 1 shows 4 input job's instances, i.e. 5 jobs, 10 jobs, 20 jobs and 50 jobs with 300 MB data size for each job. Each job tasks process continuously reduce for an expanding number of jobs in scheduler, i.e. 95 % for 5 jobs, 79 % for 10 jobs, 42 % for 20 jobs and 26 % for 50 jobs. The dynamic scheduler always remains more than 90 %. An increasing number of jobs also expand the completion time due to degrade in other scheduling approaches. Compared to other job scheduling approaches, our approach has minimum waiting time because we adding the dynamic threshold value assignment in our scheduling approach.

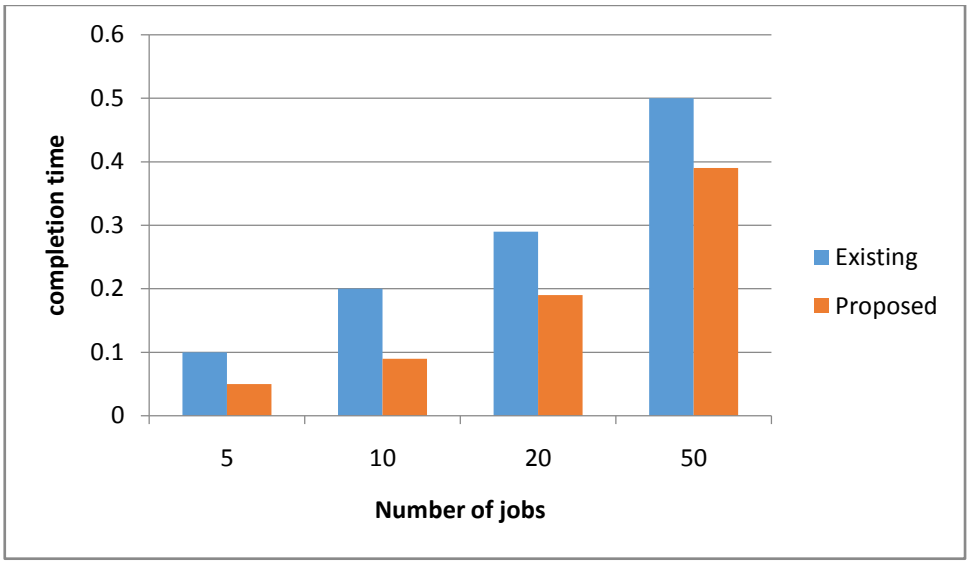

Figure 1: job execution time in queue

The job execution time in multi queue has displayed in the above figure. The dynamic multi threshold job scheduling approach takes lesser job completion time in the multi queue compared to other scheduling approaches.Working Principle: Scheduling job services over the queues of sensor node. In a ready queue, we schedule the sensed job services on a node in the middle of a number of levels. Then, a number of job services are designed in the every level of the ready queue. Before the execution of other priority jobs, the medium priority scheduler has planned to be execute jobs.

Average waiting time

We calculate the average waiting time of real rime tasks at various workloads in the following. Let us assume that p_{iji} represent the processing time of the i-thpr_i task at a node x, Where $1 \leq i \leq 3$ and $1 \leq j_i \leq n_i$.

The total processing time, $pr_i(t) = \sum_{ji=1}^{ni} piji(t)$ let us take k as the total number of levels, and the length of a time slot at the level l_j as t(j).

For real time tasks, i=1 (i.e. p1),Assuming that real time and emergency tasks rarely occur and need a very short time to get executed, pr1(t) < t(k). But all tasks, $1 \leq i1 \leq n1$, in the p1 queue complete processing and the tasks in the pr2 and pr3 queues are executed for the remaining, t2(k)= t(k)- pr1(t), period of time.

Since pr1 tasks are executed as FCFS, the average waiting time for real time, pr1 task at node x is

Average waiting Time p1(t)= $(\sum_{ji=1}^{n1-1} \sum_{m=1}^{ji} p1, m(t)) \div k1$ ----------(3)

Where the first pr1 task has no waiting time and waiting time for the j-th pr1task is equal to $\sum_{l=1}^{j} p1, m(t)$.

We processed various kind of jobs simultaneously on the nodes and examines the results with more than one parameters. We identified that both FIFO and fair share scheduling attacks very formal on our algorithm. That's why, in our experiment we took single instance to consider the scheduling algorithms.

We decrease the average time of job completion of tasks around 65%. For higher system loads all the metrics display that our approach works even better. The explanation is that if the load is higher, job scheduling is more important because more jobs are running there in the system.

The proposed approach causes minimum waiting time compared to the existing approach. Figure 2 shows the waiting time of tasks among a number levels. The proposed DMTJS scheme satisfying our expectation by giving outperforms compared to the Multilevel Queue scheduler and the existing FCFS. The proposed scheduling scheme provides the highest priority to small jobs and to preempt the emergency jobs, it also allows emergency medium queue in the each queue because all job tasks have minimum waiting time.

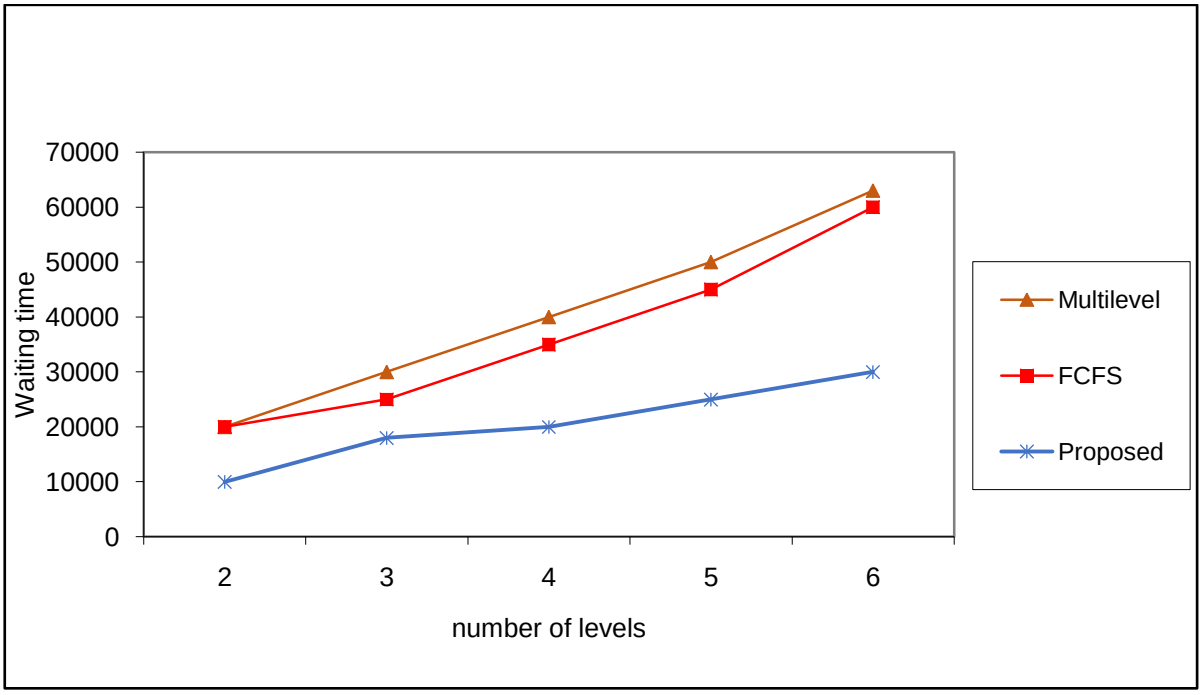

Figure 2: Waiting time of jobs in the queue

According to the increment of jobs, the time of completion also increased linearly in modified framework. The data locality also improved in our algorithm so our algorithm increments the job execution time, response time and the throughput.

References

[1] Rudraneel Chakraborty, Shikharesh Majumdar, "A Priority Based Resource Scheduling Technique for Multitenant Storm Clusters ", Dept. of Systems and Computer Engineering.

[2] Tim van der Lee, Antonio Liotta, Georgios Exarchakos," Time-scheduled Network Evaluation based on Interference ",2018 IEEE International Conference on Cloud Engineering.

[3] Zhiming Hu, Baochun Li, Zheng Qin, Rick SiowMong Goh," Job Scheduling without Prior Information in Big Data Processing Systems ",2017 IEEE 37th International Conference on Distributed Computing Systems.

[4] Zhaocong Wen, Chao Zhang,Junmin Wu, Jintao Mo1, "Research on Mixed Tasks Scheduling in YARN ",2017 IEEE.

[5] AnilkumarKothalil Gopalakrishnan, "A Preemptive Job scheduler Based on a Back propagation Neural Network ", 2017 International Conference on Computer, Communications and Electronics (Comptelix) Manipal University Jaipur, Malaviya National Institute of Technology Jaipur & IRISWORLD, July 01-02, 2017.

[6] S.Vanithamani, N.Mahendran, "Performance Analysis of Queue Based Scheduling Schemes in Wireless Sensor Networks ", Department of ECE

[7] Narayanan Sadagopan,Bhaskar Krishnamachari, "Delay efficient sleep scheduling in wireless sensor network ", Conference Paper in Proceedings - IEEE INFOCOM • January 2005.

[8] Jayasudha.J, T.Prabakaran,"Reduction of End To End Delay Using Priority Queues in Wireless Sensor Network ", International Journal of Innovative Research in Computer and Communication Engineering.Vol.2, Special Issue 1, March 2014.

[9] JinweiLiu,Haiying Shen, "Dependency-aware and Resource-efficient Scheduling for Heterogeneous Jobs in Clouds ", 2016 IEEE 8th International Conference on Cloud Computing Technology and Science.

[10] Andrei Voinescu, Dan S¸tefanTudose, Nicolae T¸apus, "Task Scheduling in Wireless Sensor Networks ", 2010 Sixth International Conference on Networking and Services.

[11] Ines Khoufi, Pascale Minet, BadrRmili, "Scheduling transmissions with latency constraints in an IEEE 802.15.4e TSCH network ", 2017 IEEE.

[12] Matthias Vodel, Mirko Lippmann, Mirko Caspar and Wolfram Hardt, "A Capable, High-Level Scheduling Concept for Application-Specific Wireless Sensor Networks ", 2010 IEEE.

[13] Jianting Yue, Dongmei Zhao, and Terence D. Todd, "Cloud Server Job Selection and Scheduling in Mobile Computation Offloading ",Globecom 2014 - Wireless Networking Symposium.

[14] Swati D. Kadu, Vivek S. Deshpande, "Characterization of Throughput in Wireless Sensor Network for MAC and Routing protocol ",2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies.

[15] Eun-Mook Lee, Ali Kashif , Dong-Hyun Lee, In-Tae Kim, Myong-Soon Park t, "Location Based Multi-Queue Scheduler in Wireless Sensor Network ", Feb. 7-10, 2010 ICACT 2010.

[16] M.-Y. Hsieh, "Data aggregation model using energy-efficient delay scheduling in multi-hop hierarchical wireless sensor networks ", Green Technologies for Wireless Communications and Mobile Computing,2011, Vol. 5, Iss. 18, pp. 2703–2711.

[17] Sami J. Habib and PaulvannaNayakiMarimuthu, "Scheduling Sensors' Tasks with Imprecise Timings within Wireless Sensor Networks ", Computer Engineering Department, 2010.